Neural Ordinary Differential Equations and Its Extensions

Luong Thuy Chung
Nguyen Nhat Mai

Supervised by
Dr. Vu Khac Ky

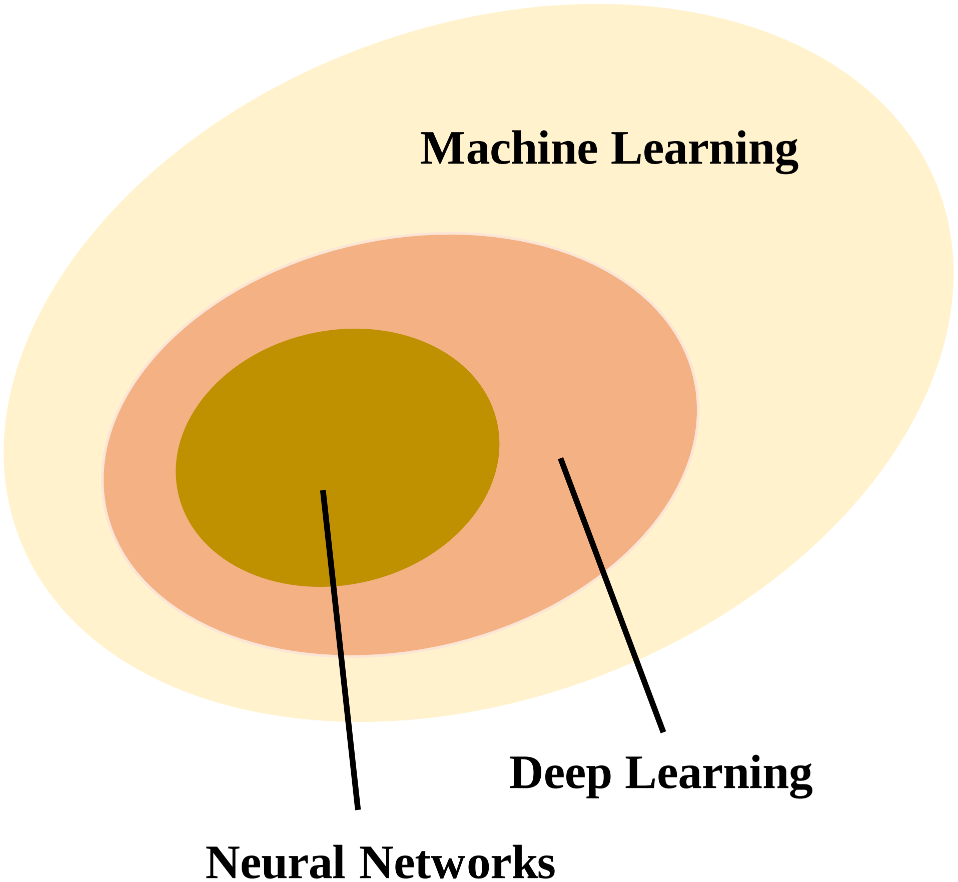
Outline

01

Introduction

Introduction to Neural ODEs-Net,
a new family of Neural Networks.

- Through years, neural networks is deeper, that the deeper the network is, the more difficult the model can learn.
- ResNets were born as the development of neural networks, that the number of classes has increased dramatically.
- In 2018, Chen et al. launched the Neural ODEs-Net, a new family of neural network.



Venn diagram showing the relationship among Machine Learning, Deep Learning and Neural Networks

02

Background

- Ordinary Differential Equations
- Neural Networks

A n^th order **Initial-Value Problems** (IPVs) includes two parts:

- A n^th order ordinary differential equation in the form of

$$y^{(n)} = f(t, y, y', y'', y^{(3)}, ..., y^{(n-1)})$$

- Initial conditions of *y* and its derivatives at a particular point of *x*:

$$\begin{cases} y(t_0) & = y_0 \\ y'(t_0) & = y_1 \\ y''(t_0) & = y_2 \\ & \dots \\ y^{(n-1)}(t_0) & = y_{n-1} \end{cases}$$

The **first-order Initial-Value Problems:**

$$y'(t) = f(t, y(t)), \qquad y(t_0) = y_0.$$

Definition 2.2.1. A function $f(t, y)$ satisfies a **Lipschitz condition** on a set D if there is a constant $L \geq 0$ such that

$$y'(t) = f(t, y), \; t_0 \leqslant t \leqslant T, \; y(t_0) = y_0$$

wherever $(t, y_1), (t, y_2)$ are in D.

The Existence and Unique Theorem for First-Order Ordinary Differential Equations. Let $f(t, y)$ is continuous on $D = \{(t, y) \mid t_0 \leq t \leq T$ and $-\infty \leq y \leq \infty\}$. If f satisfies a Lipschitz condition on D in the variable y, then the initial-value problem

$$y'(t) = f(t, y), \; t_0 \leqslant t \leqslant T, \; y(t_0) = y_0$$

has a unique solution $y(t)$ for $t \in [t_0, T]$.

Numerical Methods for Initial-Value Problems:

With initial condition $w_0 = y_0$ for each $i = 0, 1, 2, \ldots, N-1$,

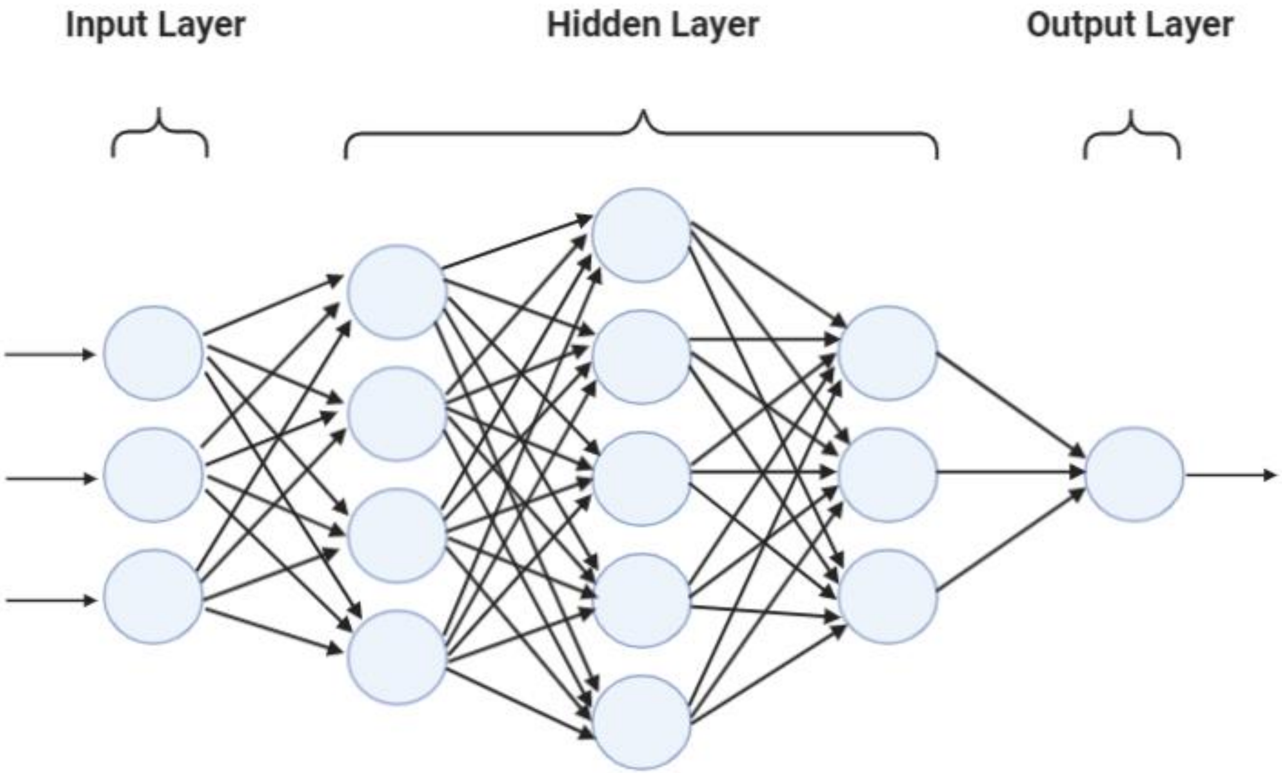
Euler's Method: $w_{i+1} = w_i + hf(t_i, w_i),$

Beside Euler's Methods, it is known that the family of **Taylor methods**, the family of **Linear Multistep Method**, or the family of **Runge-Kutta Methods** is numerical methods for approximating the solutions of the initial problems

Feedforward Neural Networks

Goal: To approximate some functions $f(.)$ that map the input x to the output $\hat{y}$ which is close to the desired value y

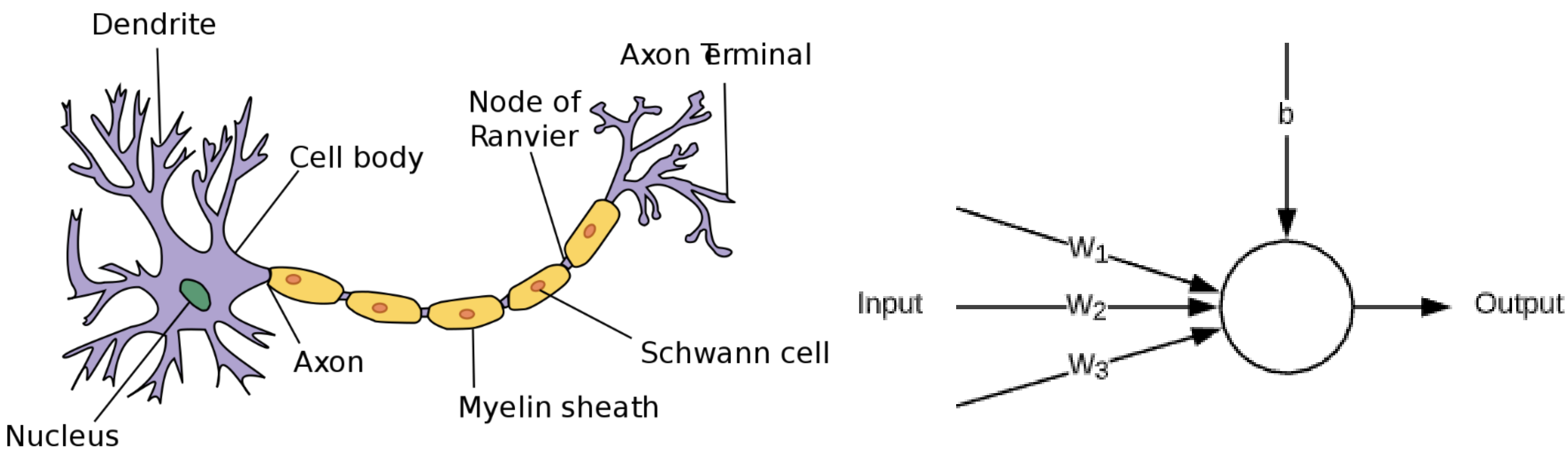
Strategies: To learn the value of parameters $\boldsymbol{W}$ and $\boldsymbol{b}$ that shows the best approximation of $f(.)$



A feedforward neural network.

Units

$$\boldsymbol{a} = \boldsymbol{W}^\top \boldsymbol{x} + \boldsymbol{b}$$



(a) (b)

(a) A human neuron[1]*, (b) A unit in neural network.*

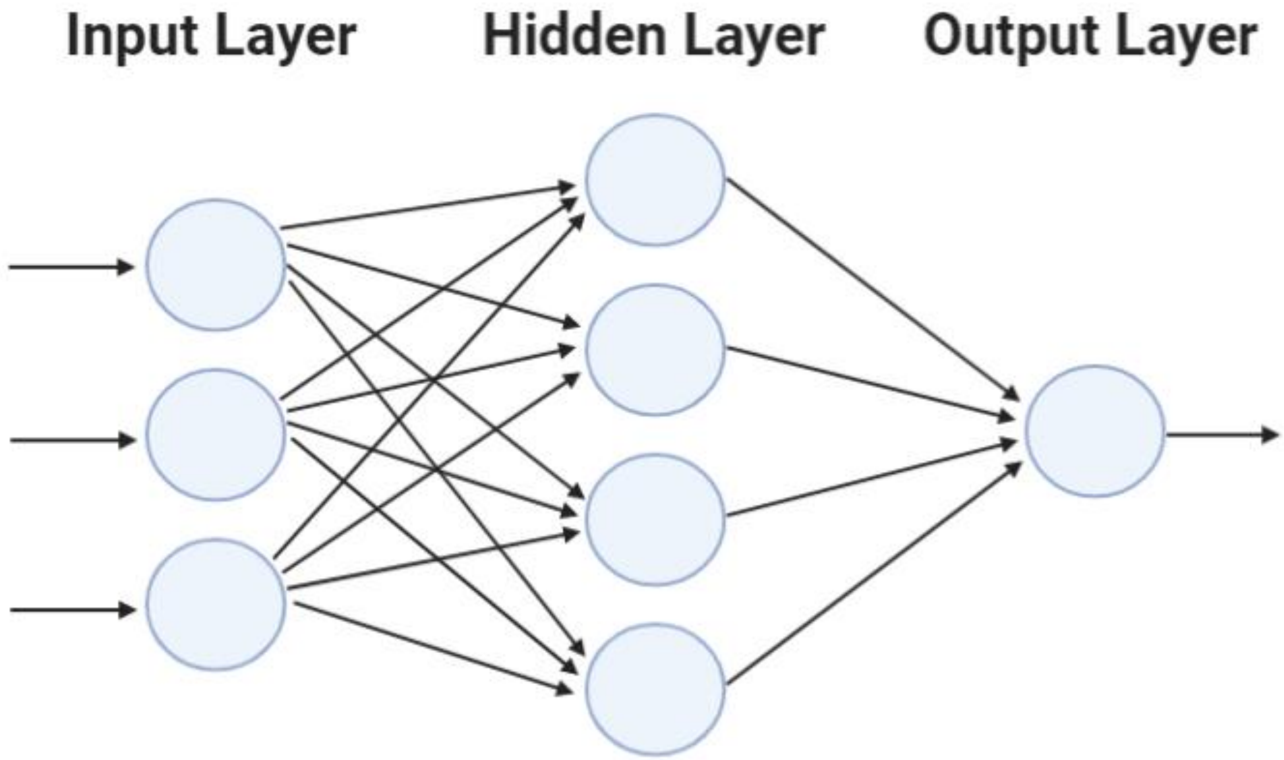
Activation Functions

- Sigmoid Function: $g(z) = \sigma(z) = \dfrac{1}{1+e^{-z}}$
- ReLU: $g(z) = max\{0, \boldsymbol{z}\}$

[1] https://simple.wikipedia.org/wiki/Neuron

Layers

A feedforward neural network consists of an input layer, an output layer, and zero or more hidden layers



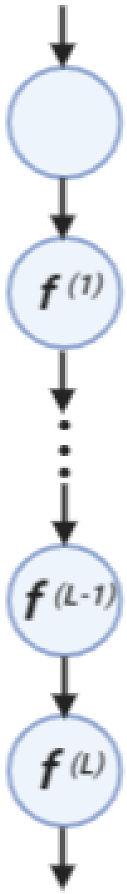
Layers in a feedforward neural network.

Architecture of Feedforward Neural Networks

Hidden state of a feedforward neural network is given by

$$\boldsymbol{h}^{(k)} = f^{(k)}(\boldsymbol{W}^{(k)\top}\boldsymbol{h}^{(k-1)} + \boldsymbol{b}^{(k)})$$

Layers in a feedforward neural network compose each other, so its architecture is named chain structure.



Architecture of Feedforward Neural Networks.

The Universal Approximation Theorem

It is shown that there exists a feedforward neural network which is large enough to represent any functions.

It is expected that the network has more layers, it can produce the output closer to the desired valued.

Gradient-Based Optimization:

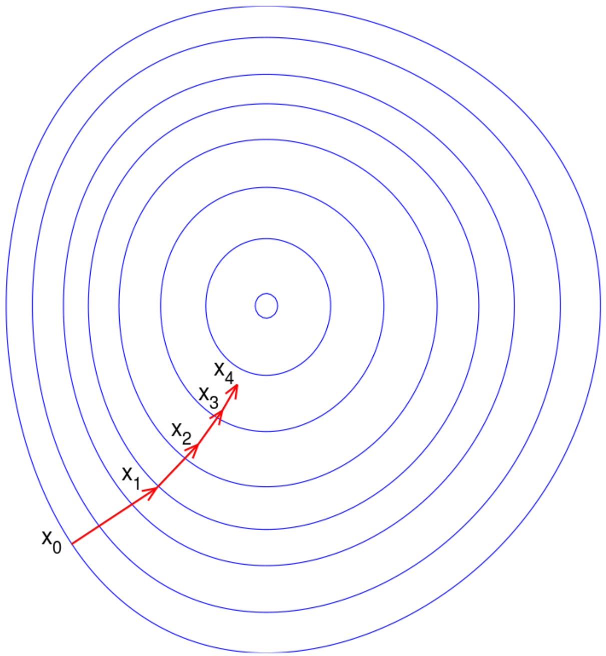
Cost Function: $J(\boldsymbol{W}, \boldsymbol{b}) = \frac{1}{m} \sum_{i=1}^{m} L(\hat{y}_i, y_i)$

Steepest Gradient Descent:

Parameters Update

$$\boldsymbol{W} \leftarrow \boldsymbol{W} - \alpha \nabla_{\boldsymbol{W}} f(\boldsymbol{W}),$$
$$\boldsymbol{b} \leftarrow \boldsymbol{b} - \alpha \nabla_{\boldsymbol{b}} f(\boldsymbol{b}),$$

where, α is learning rate.



Steps of Gradient descent[2].

Stochastic Gradient Descent (SGD):

Instead of training all large dataset, in SGD, Dataset is sampled into minibatch size m'.

$$\boldsymbol{W} \leftarrow \boldsymbol{W} - \alpha \boldsymbol{g}_{\boldsymbol{W}} f(\boldsymbol{W}),$$
$$\boldsymbol{b} \leftarrow \boldsymbol{b} - \alpha \boldsymbol{g}_{\boldsymbol{W}} f(\boldsymbol{b}),$$

where, α is learning rate.

[2] https://en.wikipedia.org/wiki/Gradient_descent

Learning Process

Forward Propagation:

Input: $\boldsymbol{h}^{(0)} = \boldsymbol{x}$

Output of each layer: $\boldsymbol{h}^{(k)} = f^{(k)}(\boldsymbol{W}^{(k)\top}\boldsymbol{h}^{(k-1)} + \boldsymbol{b}^{(k)})$

where, $\boldsymbol{h}^{(k)}$ is hidden state of the k^{th} layer

Output of Forward Propagation Process:

$$\hat{\boldsymbol{y}} = f^{(L)}(\boldsymbol{W}^{(L)\top} f^{(L-1)}(\ldots f^{(1)}(\boldsymbol{W}^{(1)\top}\boldsymbol{x} + \boldsymbol{b}^{(1)})\ldots) + \boldsymbol{b}^{(L)})$$

Learning Process

Backward Propagation:

Cost function: $$J(\boldsymbol{W}, \boldsymbol{b}) = \frac{1}{m} \sum_{i=1}^{m} L(\hat{y}_i, y_i)$$

Update parameters ***W*** and ***b*** with Stochastic Gradient Descent:

$$\begin{aligned} \boldsymbol{W} &\leftarrow \boldsymbol{W} - \alpha \boldsymbol{g}_{\mathbf{W}} f(\boldsymbol{W}), \\ \boldsymbol{b} &\leftarrow \boldsymbol{b} - \alpha \boldsymbol{g}_{\mathbf{W}} f(\boldsymbol{b}), \end{aligned}$$

where, α is learning rate.

Vanishing Gradient Problem

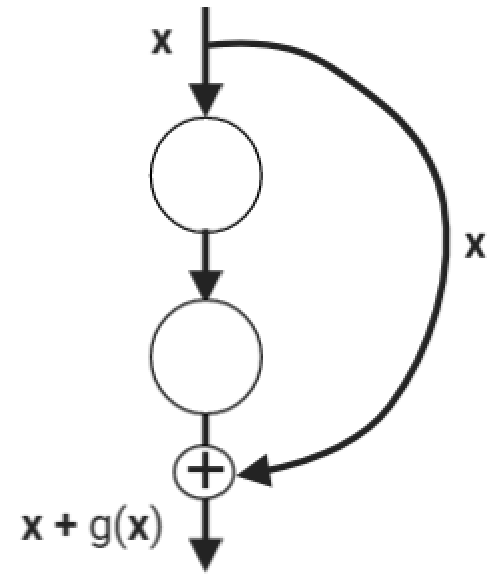
Parameters update not too much when vanishing gradient occurs. Therefore, our model cannot effectively learn.

Residual Neural Network

Hidden state in a residual neural network is given by

$$z^{(i+1)} = z^{(i)} + g(z^{(i)}; \theta^{(i)})$$

where, $z^{(i)}$ is hidden state at the i^{th} layer.



A building block in a residual network

03

Neural Ordinary Differential Equations

- What is Neural ODEs-Net?
- Learning Process of Neural ODEs-Net
- Implementation for Supervised Learning Problems
- Benefits of Neural ODEs-Net

ResNets

Since, the hidden state of residual neural network,

$$\mathbf{z_{t+1}} = \mathbf{z}_t + g(\mathbf{z}_t, \theta_t)$$

we have,

$$\frac{\mathbf{z}_{t+1} - \mathbf{z}_t}{(t+1) - t} = g(\mathbf{z}_t, \theta_t)$$

Neural ODEs-Net

Adding more layers until it goes to **infinity**, then we get following IVP:

$$\frac{\mathrm{d}\mathbf{z}(t)}{\mathrm{d}t} = g(\mathbf{z}(t), \theta(t)), \quad t \in [0, T]$$

$$\mathbf{z}(0) = \mathbf{x}$$

Using a neural network of form $f(z(t), t, \theta)$ to replace $g(.)$:

$$\frac{\mathrm{d}\mathbf{z}(t)}{\mathrm{d}t} = f(\mathbf{z}(t), t, \theta), \quad t \in [0, T]$$

$$\mathbf{z}(0) = \mathbf{x}$$

Learning Process of Neural ODE-Net

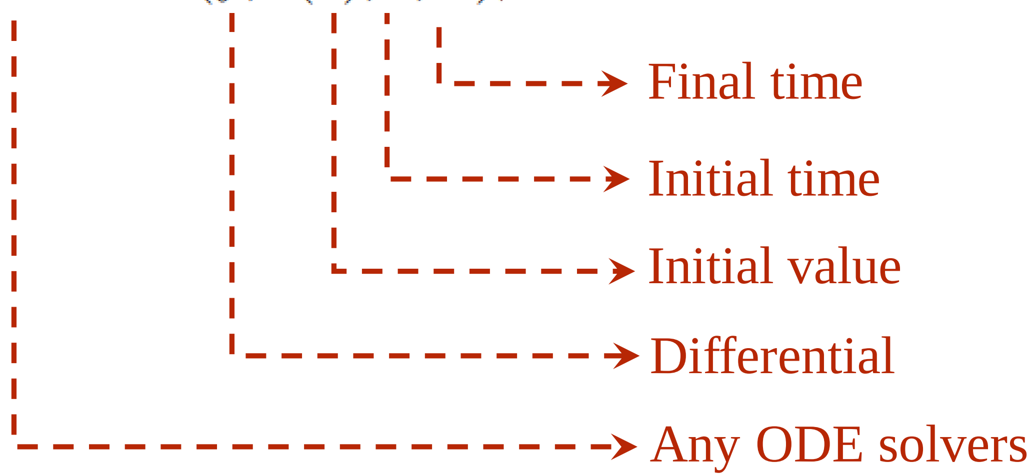
Continuous Forward Propagation:

Input: $\mathbf{z}(0) = \mathbf{x}$

Output: $\mathbf{z}(T) = \mathbf{z}(0) + \int_{0}^{T} f(\mathbf{z}(t), t, \theta)\mathrm{d}t$

Forward Propagation:

$$\mathbf{z}(T) = \mathrm{ODESolver}(f, \mathbf{z}(0), 0, T).$$



Learning Process of Neural ODE-Net

Continuous Backward Propagation:

Loss function:

$$L(\mathbf{z}(T)) = L\left(\mathbf{z}(0) + \int_0^T f(\mathbf{z}(t), t, \theta)\mathrm{d}t\right)$$

$$\boxed{\frac{\partial L}{\partial \theta}} = ?$$

Learning Process of Neural ODE-Net

Adjoint Method:

Define: $\lambda(t) = \partial_{\mathbf{z}(t)} L,$ (Adjoint State)

$\dot{\lambda} = -\lambda(t)\partial_{\mathbf{z}} f,$ (Adjoint DiffEq)

$$\boxed{\frac{\partial L}{\partial \theta}} = -\int_0^T \lambda(t)\partial_\theta f \, dt.$$

Learning Process of Neural ODE-Net

Adjoint Method: $\lambda(t) = \partial_{\mathbf{z}(t)} L, \quad \dot{\lambda} = -\lambda(t)\partial_{\mathbf{z}} f$

Forward: $\mathbf{z}(T) = \text{ODESolver}(f, \mathbf{z}(0), 0, T)$ => $\boxed{\lambda(T) = \partial_{\mathbf{z}(T)} L}$

Learning Process of Neural ODE-Net

Adjoint Method: $\lambda(t) = \partial_{\mathbf{z}(t)} L, \quad \dot{\lambda} = -\lambda(t)\partial_{\mathbf{z}} f$

Forward: $\mathbf{z}(T) = \text{ODESolver}(f, \mathbf{z}(0), 0, T)$ => $\lambda(T) = \partial_{\mathbf{z}(T)} L$

Backward: $\mathbf{z}(0) = \text{ODESolver}(f, \mathbf{z}(T), T, 0)$

$$\lambda(0) = \text{ODESolver}(-\lambda(t)\partial_{\mathbf{z}} f, \lambda(T), T, 0)$$

$$\partial_{\theta} L = \text{ODESolver}(-\lambda(t)\partial_{\theta} f, \mathbf{0}_{|\theta|}, T, 0)$$

Learning Process of Neural ODE-Net

Adjoint Sensivity Method: $\lambda(t) = \partial_{\mathbf{z}(t)} L, \quad \dot{\lambda} = -\lambda(t)\partial_{\mathbf{z}} f$

Forward: $\mathbf{z}(T) = \text{ODESolver}(f, \mathbf{z}(0), 0, T)$

Backward: $\mathbf{z}(0) =$

$\lambda(0) =$

$\partial_\theta L =$

Combine 3 ODE Solves into 1

ODESolver
(DiffFunc, Initial Value, Start Time, End Time)

Learning Process of Neural ODE-Net

Adjoint Sensivity Method: $\lambda(t) = \partial_{\mathbf{z}(t)} L, \quad \dot{\lambda} = -\lambda(t)\partial_{\mathbf{z}} f$

Forward: $\mathbf{z}(T) = \text{ODESolver}(f, \mathbf{z}(0), 0, T)$

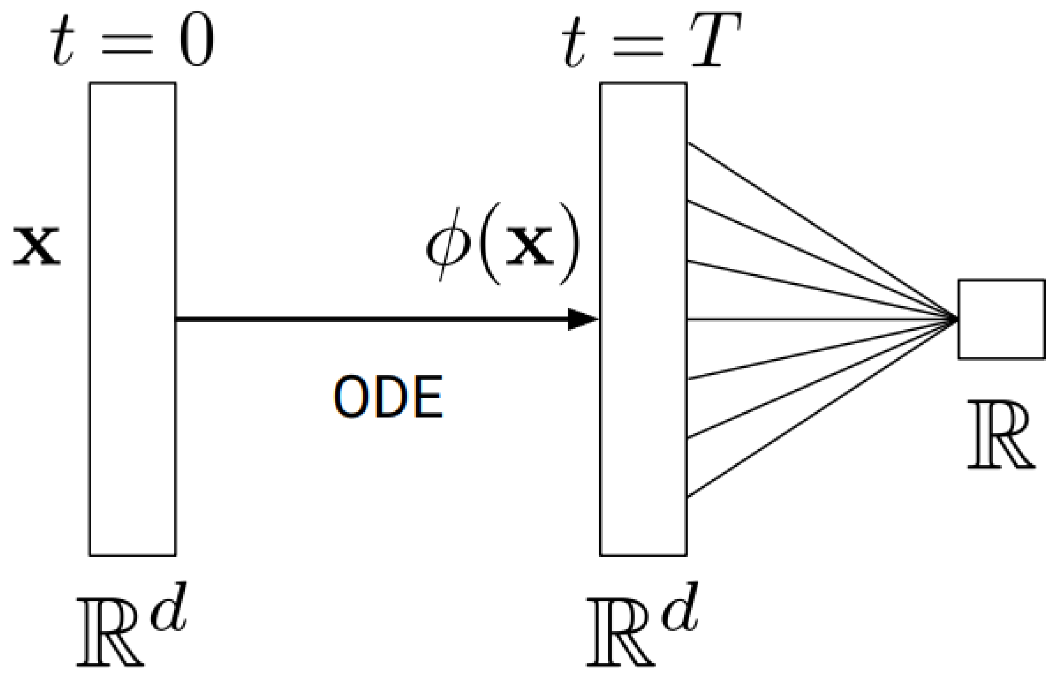
Backward:

$$\begin{bmatrix} \mathbf{z}(0) \\ \lambda(0) \\ d_\theta L(\mathbf{z}(T)) \end{bmatrix} = \text{ODESolver}\left(\begin{bmatrix} f \\ -\lambda(t)\partial_z f \\ -\lambda(t)\partial_\theta f \end{bmatrix}, \begin{bmatrix} \mathbf{z}(T) \\ \lambda(T) \\ \mathbf{0} \end{bmatrix}, T, 0 \right)$$

DiffFunc Initial Value

Implementation for Supervised Learning Problems

A Neural ODEs-Net is followed by a linear layer.



Architecture of Neural ODE-Net followed by a linear layer[3]

[3] R. T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. Duvenaud. Neural ordinary differential equations. *Advances in Neural Information Processing Systems*, 2018.

Benefits of Neural ODEs-Net

Memory Benefits:

- No need to store any intermediate quantity of the forward propagation.
- The model can be trained with **constant memory**.

Computation Benefits:

- Modern ODE solvers quickly adjust their evaluation strategy to accomplish the required level of accuracy.
- The evaluating cost scales with the problem complexity

04

Extensions of Neural ODEs-Net

- Neural ODEs-Net with Evolutionary Parameters
- Neural ODEs-Net with Extra Dimensions

Property 1: **Trajectories in Neural ODEs-Net cannot intersect**

Fundamental Theorem of ODEs

$$\boldsymbol{z}(t) \text{ is a flow.}$$

ODE trajectories:

Proposition 4.1.1.

Let $\boldsymbol{z}_{\mathbf{1}}(t)$ and $\boldsymbol{z}_{\mathbf{2}}(t)$ be two trajectories of and ODE with two different initial conditions, $\boldsymbol{z}_{\mathbf{1}}(t) \neq \boldsymbol{z}_{\mathbf{2}}(t)$ for all $t \in (0, T]$. This implies that ODE trajectories do not intersect each other.

Property 2: **Neural ODEs-Net describes a Homeomorphism**

- A homeomorphism function is a continuous bijection that has a continuous inverse function.

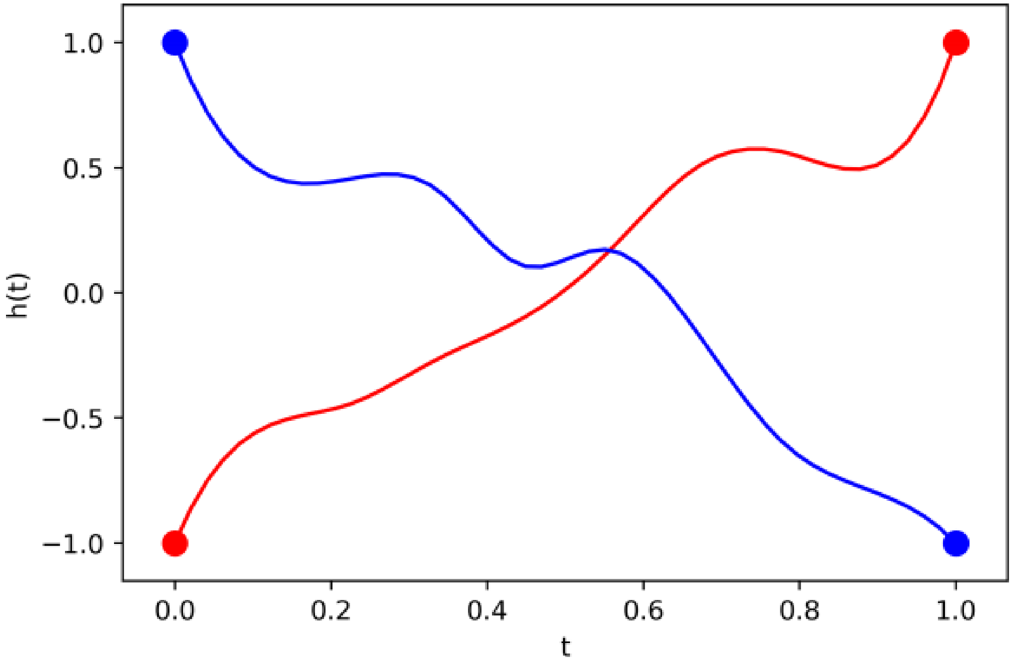
A continuous deformation between a coffee mug and a donut illustrating that they are homeomorphic.[4]

- Neural ODEs-Net describes a Homeomorphism.

[4] https://en.wikipedia.org/wiki/Homeomorphism#Properties

Functions Neural ODEs-Net cannot Represent

Let $h_{1d}: \mathbb{R} \to \mathbb{R}$ be a function such that $h_{1d}(-1) = 1$ and $h_{1d}(1) = -1$.



Continuous trajectories mapping -1 to 1 (red) and 1 to -1 (blue) must intersect each other, which is not possible for an ODE [5]

[5] E. Dupont, A. Doucet, and Y. W. Teh. Augmented neural odes. In H. Wallach, H. Larochelle, A. Beygelzimer, F. dAlch´e-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*

Functions Neural ODEs-Net cannot Represent

Not Increasing Functions in One-dimensional Space

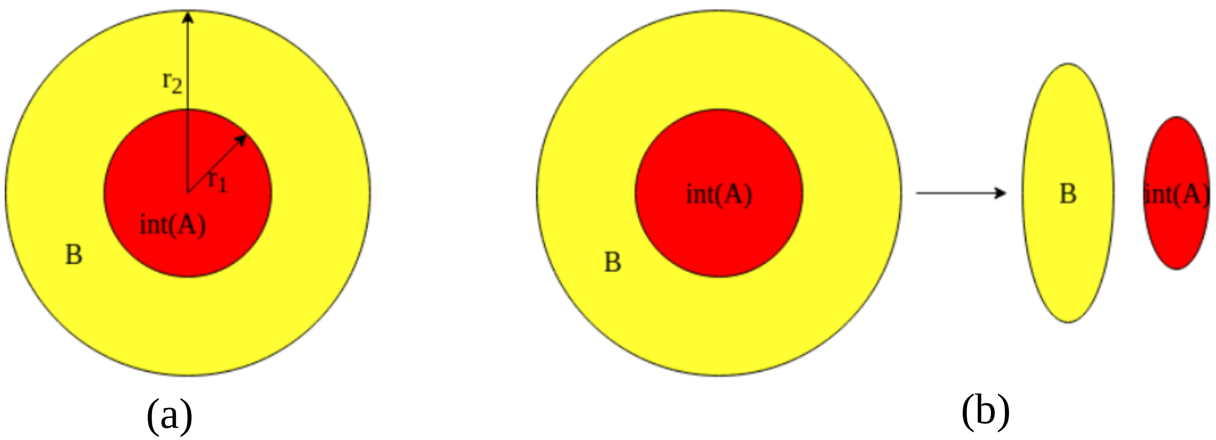
Proposition 4.2.1.

Neural ODEs-Net cannot represent **a not increasing function** $h : \mathbb{R} \to \mathbb{R}$.

Functions Neural ODEs-Net cannot Represent

Let $g(\mathbf{x}) : \mathbb{R}^d \to \mathbb{R}$ and $0 < r_1 < r_2$, such that:

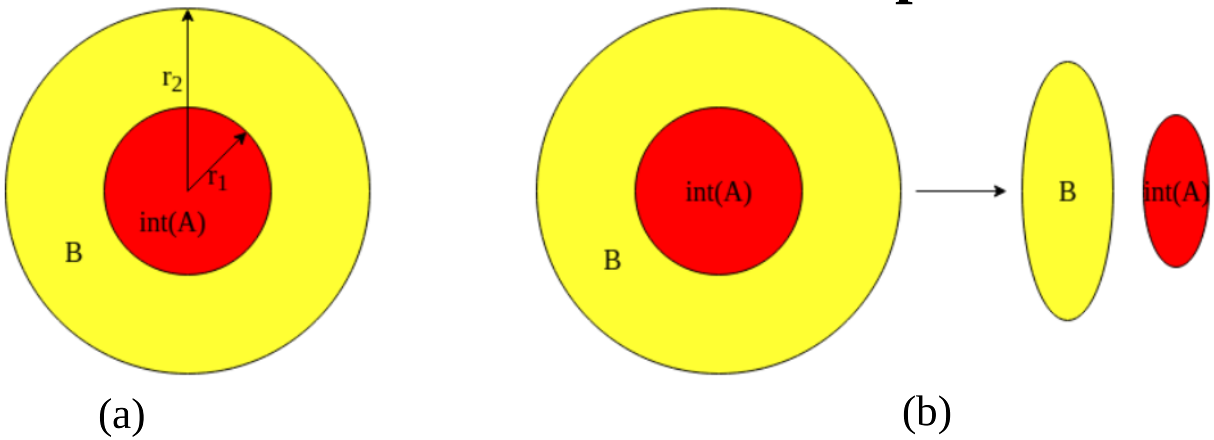
$$\begin{cases} g(\mathbf{x}) = -1 \quad if \quad ||\mathbf{x}|| < r_1 \\ g(\mathbf{x}) = 1 \quad if \quad r_1 \leq ||\mathbf{x}|| \leq r_2 \end{cases}$$



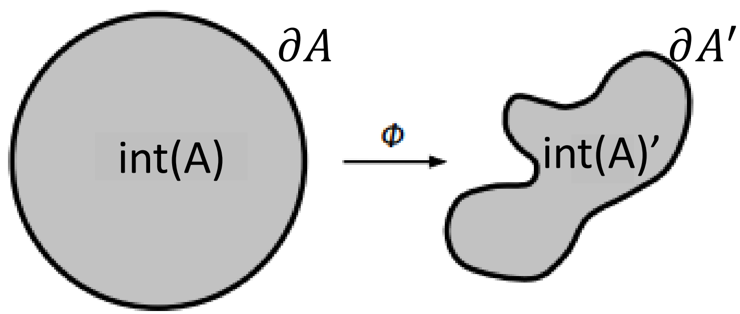
*(a) Diagram of g(**x**) in 2-dimentional space. (b) An example of the feature mapping φ(**x**) from input data to features.*

[6] E. Dupont, A. Doucet, and Y. W. Teh. Augmented neural odes. In H. Wallach, H. Larochelle, A. Beygelzimer, F. dAlch´e-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*

Functions Neural ODEs-Net cannot Represent



(a) Diagram of $g(\mathbf{x})$ in 2-dimentional space. (b) An example of the feature mapping $\phi(\mathbf{x})$ from input data to features.



An example of how ϕ transforms the disk. [6]

[6] E. Dupont, A. Doucet, and Y. W. Teh. Augmented neural odes. In H. Wallach, H. Larochelle, A. Beygelzimer, F. dAlch´e-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*

Extensions: ANODEs-Net with Extra Dimensions

In ANODEs-Net with Extra Dimensions model, we lift the original model ($\mathbb{R}^d$) up the higher dimensional space ($\mathbb{R}^{d+p}$).

$$\frac{d}{dt}\begin{bmatrix}\mathbf{z}(t)\\ \mathbf{u}(t)\end{bmatrix} = f\left(\begin{bmatrix}\mathbf{z}(t)\\ \mathbf{u}(t)\end{bmatrix}, t\right),$$

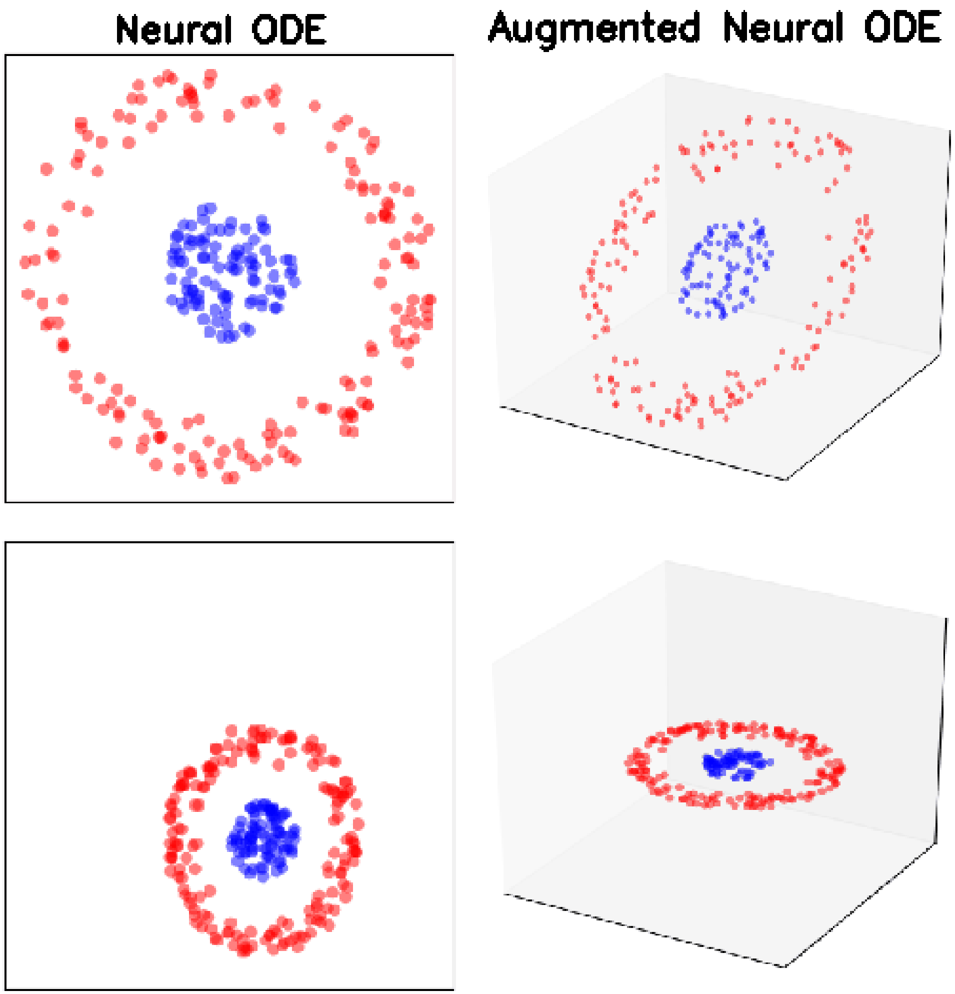
with input:

$$\begin{bmatrix}\mathbf{z}(0)\\ \mathbf{u}(0)\end{bmatrix} = \begin{bmatrix}\mathbf{x}_0\\ \mathbf{0}\end{bmatrix}$$

[4]

[4] E. Dupont, A. Doucet, and Y. W. Teh. Augmented neural odes. In H. Wallach, H. Larochelle, A. Beygelzimer, F. dAlch´e-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*

Extensions: ANODEs-Net with Extra Dimensions



[4]

[4] E. Dupont, A. Doucet, and Y. W. Teh. Augmented neural odes. In H. Wallach, H. Larochelle, A. Beygelzimer, F. dAlch´e-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*

Extensions: ANODEs-Net with Evolutionary Parameters

Neural ODEs-Net

$$\frac{d\mathbf{z}(t)}{dt} = f(\mathbf{z}(t), \theta, t)$$

θ **is fixed over time**

ANODEs-Net with Evolutionary Parameters

$$\frac{d\mathbf{z}(t)}{dt} = f(\mathbf{z}(t), \theta(t), t)$$

$\theta(t)$ **depends on** t

A coupled system of ODEs – Version 1:

$$\begin{cases} \mathbf{z}(T) = \mathbf{z}(0) + \int_0^T f(\mathbf{z}(t), \theta(t), t)dt, & \mathbf{z}(0) = \mathbf{x} \\ \theta(t) = \theta(0) + \int_0^t g(\theta(t), \omega, t)dt, & \theta(0) = \theta_0 \end{cases}$$

"Activation network"

"Weight network"

- If g = 0, then it is exactly the original Neural ODEs-Net with fixed weights

A coupled system of ODEs – Version 2:

A constrained optimization problem:

$$\min_{p, w_0} \mathcal{J}(\mathbf{z}(T)) = \frac{1}{N} \sum_{i=1}^{N} l(\mathbf{z}(T); \underline{x_i, y_i}) + \underline{R(w_0, p)}$$

(x_i, y_i) is the i^{th} training sample and its label

Regularization

A coupled system of ODEs – Version 2:

A constrained optimization problem:

$$\min_{p,w_0} \mathcal{J}(\mathbf{z}(T)) = \frac{1}{N}\sum_{i=1}^{N} l(\mathbf{z}(T); x_i, y_i) + R(w_0, p)$$

subject to

$$\frac{\mathrm{d}\mathbf{z}(t)}{\mathrm{d}t} = f(\mathbf{z}(t), \theta(t), t), \qquad \mathbf{z}(0) = \mathbf{z}_0$$

"Activation ODE"

$$\frac{\mathrm{d}w(t)}{\mathrm{d}t} = g(w(t), p, t), \qquad w(0) = w_0$$

"Evolution ODE"

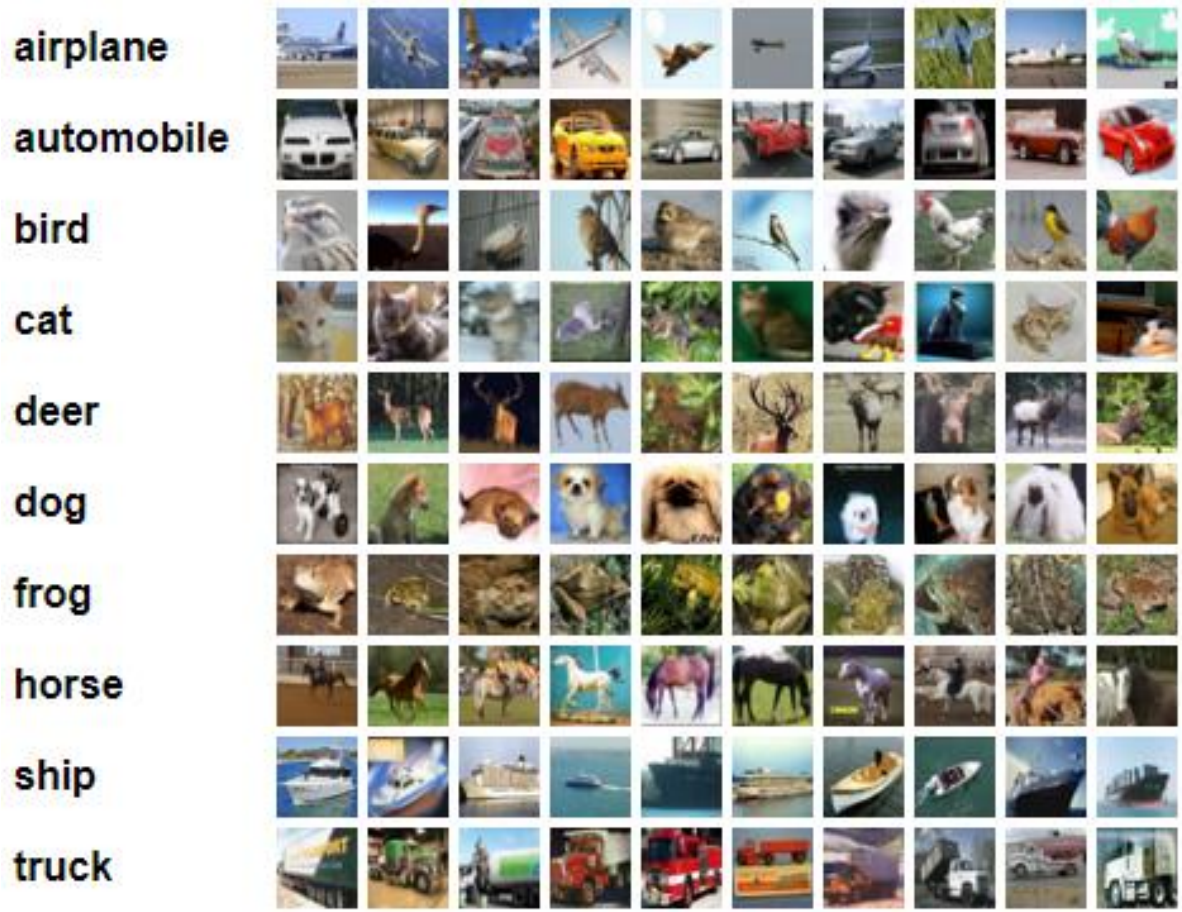
$$\theta(t) = \int_0^t K(t-\tau) w(\tau) \mathrm{d}\tau,$$

K: a convolution kernel/a Dirac delta function

05

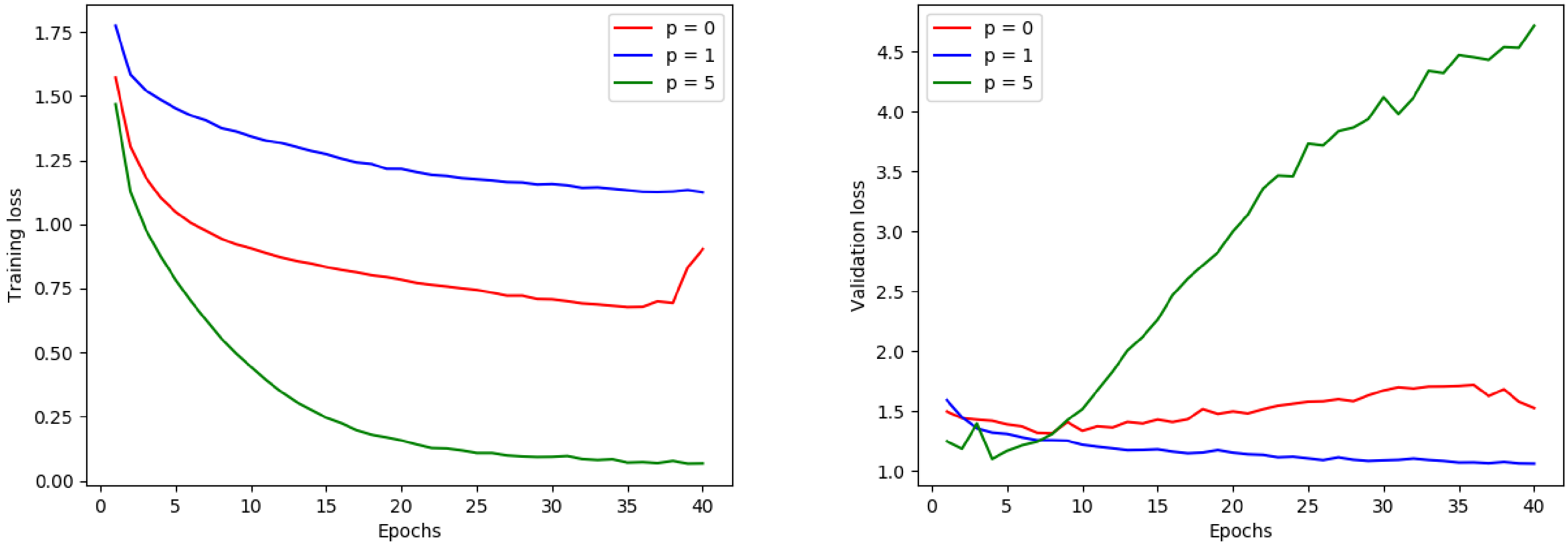
Experimental Results

- Compare training loss of pure Neural ODE-Net and its extensions
- Compare test accuracy between models



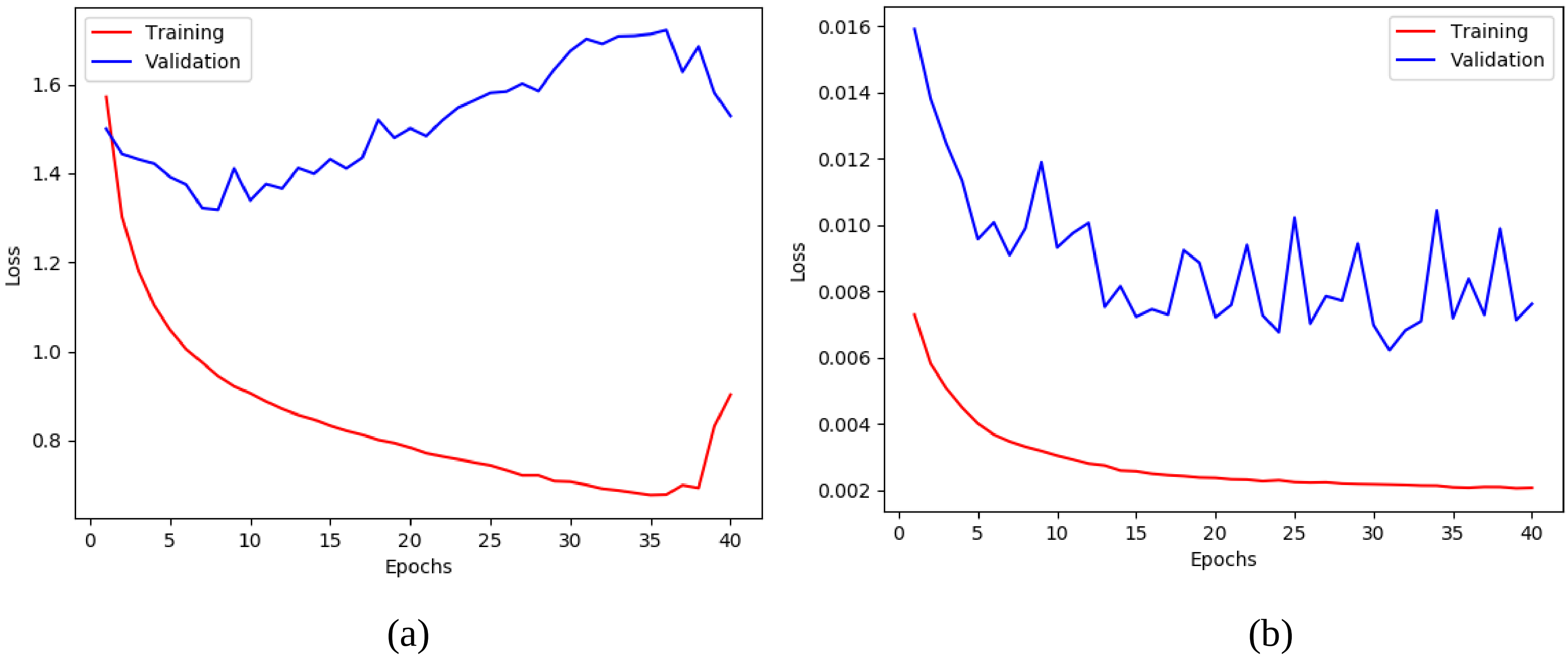
Ten classes of CIFAR-10 dataset
and ten image from each of them[5]

[5] https://www.cs.toronto.edu/~kriz/cifar.html

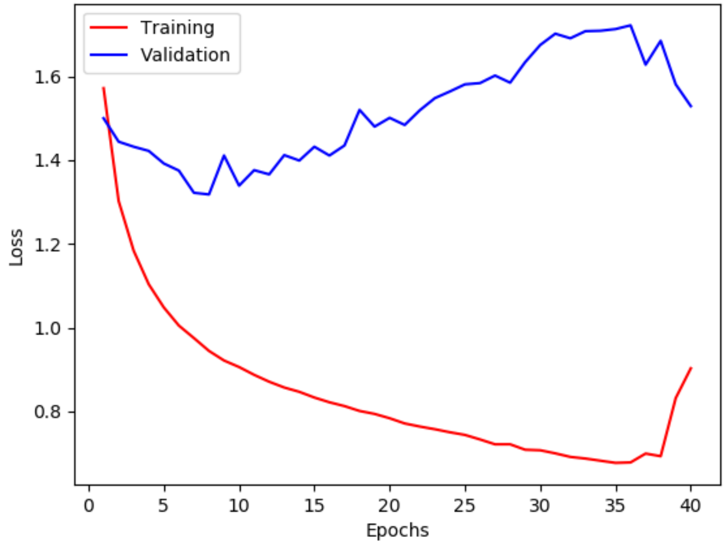


(a) (b)

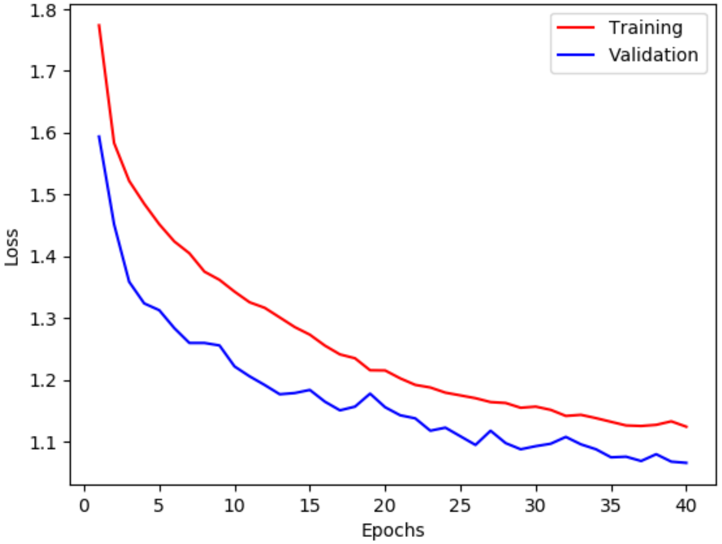
Training loss and validation loss for original model and augmented models on CIFAR-10 dataset. (a) Training losses (b) Validation losses. Note that p indicates the numbers of augmented dimensions, so p = 0 indicates the original neural ODEs-Net model.



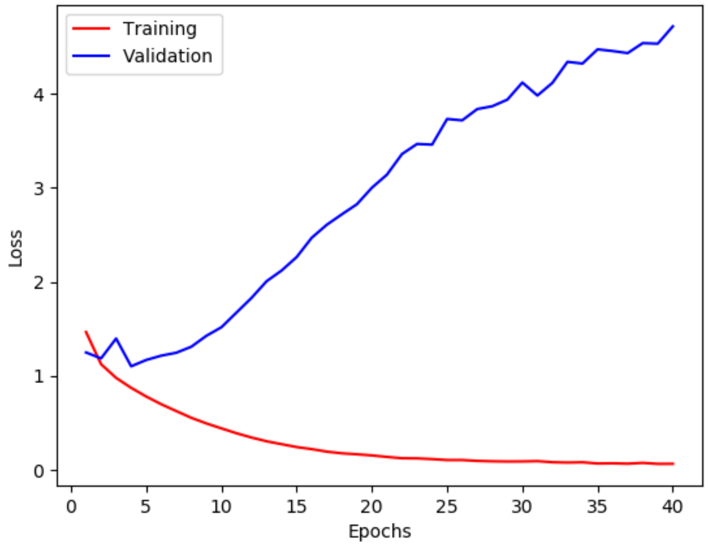
Training loss and validation loss for original model and augmented models on CIFAR-10 dataset. (a) Neural ODEs Model (b) NODEs with Evolutionary Parameters Model



(a)



(b)



(c)

Training and validation losses for models (a) The original NODEs (b) NODEs with extra dimensions $p = 1$ *(c) NODEs with extra dimensions* $p = 5$

	Min	Max	Average
Original Neural ODEs-Net	32.81%	65.63%	51.76%
NODEs-Net with Extra Dimensions $p = 1$	56.25%	69.92%	62.27%
NODEs-Net with Extra Dimensions $p = 5$	40.63%	71.88%	55.45%
NODEs-Net with Evolutionary Parameters	76.92%	77.45%	77.30%

Test accuracies for NODEs model and its extension

06

Conclusion & Future Works

- Conclusion
- Future works

Conclusion

- Recalled the knowledge of ordinary differential equations and feedforward neural networks.
- Introduced Neural ODEs-Net which consists of its architecture, learning process and how to apply it for a supervised learning problems.
- Pointed out properties of Neural ODEs-Net, its strengths and weakness.
- Mentioned two extensions of Neural ODEs-Net with extra dimensions and evolutionary parameters.
- Experimented with Neural ODE models and received the positive results.

The training time:

Training time of a neural ODEs model is quite high compared to residual neural network. However, it is proved that it is possible to decrease the training time of a neural ODEs model[6].

The representation ability:

Neural ODEs-Net is not an universal approximation. A new promising result which is proved that it is universal approximation was introduced in 2020 with providing additional theoretical results[7].

[6] C. Finlay, J.-H. Jacobsen, L. Nurbekyan, and A. M. Oberman. How to train your neural ode: theworld of jacobian and kinetic regularization, 2020.
[7] P. Kidger, J. Morrill, J. Foster, and T. Lyons. Neural controlled differential equations for irregulartime series, 2020.

Thank you!

for your attendance

Q&A